THE USE OF REDUNDANCY
TO IMPROVE PERFORMANCE
IN AN INVERTED HIERARCHICAL DATA STRUCTURE

John J. Donovan *
Seymour N. Small **

June 1980

CISR No. 58
Sloan WP No. 1145-80

THE USE OF REDUNDANCY TO IMPROVE PERFORMANCE IN AN INVERTED HIERARCHICAL DATA STRUCTURE

John J. Donovan *
Seymour N. Small **

June 1980

CISR No. 58
Sloan WP No. 1145-80

* Massachusetts Institute of Technology

** Center for Birth Defects Information Services
Tufts-New England Medical Center

CONTENTS

Introduction 3

Background Concepts in Hierarchical Data Bases 4

Improving Performance with Family Redundancy 7

Quantification of Savings Potential Using Family Redundancy 13

Improving Performance with Collapsing of Levels 15

Quantification of Savings Potential Using Collapsing of Levels . . . 17

Case Study 18

Restructuring the BDIS Database 24

Quantification of Savings Potential for BDIS 27

Experimental Results 30

Other Issues to be Considered 34

Conclusion 35

References 36

Introduction

This paper suggests two techniques for reducing the costs of operating certain database applications that use a hierarchical structure with inversion on low-level fields. These techniques are:

1) introducing "family redundancy," i.e., introducing redundant information into a record that was obtained directly, or through a summarization process, from its "parent," "child," or "sibling" records, without altering the structure of the hierarchy, or eliminating fields from any records.

2) incorporating one record level into another.

The paper contains formulas which make explicit the costs and savings associated with each technique. A case study will be presented in which the formulas are used to predict possible savings. The observed performance improvements, following the implementation of these techniques, are reported. These improvements corresponded to those predicted by the formulas.

The case study chosen is a birth defects diagnostic assist system. For this system, these techniques reduced the average costs by approximately eighty per cent.

Background Concepts in Hierarchical Data Bases

The hierarchical model of data is extensively used in a broad number of applications*. For certain applications, those characterized by a relatively stable database with pre-defined queries or operations, we have found the hierarchical system to be quite appropriate [4].

A key goal of a system designer, in general, is to reduce data redundancy, since redundancy wastes space and creates the potential for inconsistency in the data. However, redundancy can also yield significant improvement in access times, and this savings must be weighed against the corresponding costs**. In this paper, two general redundancy-based database design techniques are introduced. In order to illustrate these techniques, consider the following hypothetical database:

Figure 1A depicts a hierarchical structure of three record types:

* STUDENT - containing ID number, name and address.
* YEAR - containing year and the department in which the student was registered during that year.
* COURSE - containing course number and the grade received by the student.

Figure 1B depicts one instance of that hierarchy in which the student is Steven Melnick, and his address is Eastgate. He has attended school

----

* IMS, a hierarchical data base management system, is reported as having the second most number of installations in the world [7].

** See Severance for a general model of file costs [9].

for two years. In 1978, he was in Department 6, and in 1979, he switched to Department 15. He took two courses during 1978, and one in 1979.

This database organization is particularly well-suited for responding to queries such as: "Print all grades for Melnick." Assuming there is an index on the name field, Melnick's STUDENT record could be accessed directly, and all of the YEAR and COURSE records read sequentially.

```
STUDENT

            +--------------------------------+
            |            |       |           |
            | ID_NUMBER* | NAME  |  ADDRESS  |
            |            |       |           |
            +--------------------------------+
                           |
YEAR                       |
                           |
            +--------------------------------+
            |              |                 |
            |    YEAR*     |     DEPT        |
            |              |                 |
            +--------------------------------+
                           |
COURSE                     |
                           |
            +--------------------------------+
            |              |                 |
            |   COURSE*    |   GRADE         |
            |              |                 |
            +--------------------------------+
```

Figure 1A: Sample Hierarchy (* = inverted field)

```
            +--------------------------------------------+
            |          |                  |              |
            | 123-456  | Steven Melnick   |   Eastgate   |
            |          |                  |              |
            +--------------------------------------------+
                     |                        |
            +-------------------+      +-------------------+
            |        |          |      |         |         |
            |  1978  |    6     |      |  1979   |   15    |
            |        |          |      |         |         |
            +-------------------+      +-------------------+
                 |           |                 |
          +-----------+  +-----------+     +-----------+
          |        |  |  |        |  |     |        |  |
          | 15.564 | A|  | 15.565 | B|     | 15.571 | A|
          |        |  |  |        |  |     |        |  |
          +-----------+  +-----------+     +-----------+
```

Figure 1B: Instance of sample hierarchy

Improving Performance with Family Redundancy

Let us examine the steps involved in answering the following, more difficult query: "List all students who received a grade point average (GPA) greater than 3.4 in 1979."

Assuming that the YEAR field is indexed, one access would be required to retrieve the 1979 YEAR record for the first student enrolled that year; then one access would be required for each of the COURSE records underneath that YEAR record. Using the grades retrieved, one student's GPA for 1979 would then be computed. If the average exceeded 3.4, an access would be made upward from the YEAR record to the parent STUDENT record to get the name of the student. These steps would be repeated for every student who was enrolled in 1979. Thus, it would be necessary to access the middle of the hierarchy, go downward, and if the query condition was met, return to the root record.

In order to avoid accessing all of the 1979 COURSE records for each student, the grade point average could be computed and placed in the YEAR record, as illustrated in Figures 2A and 2B. Under this scheme, a maximum of two accesses per student are required to answer the query, and the COURSE records need never be accessed. Thus, all downward accesses have been eliminated, and it is now only necessary to access the middle of the hierarchy, and if the query was met, return to the root record.

However, redundancy has been introduced, resulting in extra storage and maintenance. For example, if Melnick's grade in course 15.565 is changed to an 'A', then both the COURSE record and the YEAR record must be updated. In the structure of Figure 1A, on the other hand, only the COURSE record need be updated.

```
STUDENT

            +----------------------------------+
            |               |       |          |
            |  ID_NUMBER*   |  NAME |  ADDRESS |
            |               |       |          |
            +----------------------------------+
                             |
YEAR                         |
                             |
            +----------------------------------+
            |          |           |           |
            |  YEAR*   |   DEPT    |    GPA    |
            |          |           |           |
            +----------------------------------+
                             |
COURSE                       |
                             |
            +----------------------------------+
            |                |                 |
            |     COURSE*    |   GRADE         |
            |                |                 |
            +----------------------------------+
```

Figure 2A: Placing GPA in YEAR

```
  +----------------------------------------------+
  |           |                   |              |
  | 123-456   |  Steven Melnick   |   Eastgate   |
  |           |                   |              |
  +----------------------------------------------+
            |                          |
  +----------------------+    +----------------------+
  |      |         |     |    |      |        |      |
  | 1978 |    6    | 3.5 |    | 1979 |   15   | 4.0  |
  |      |         |     |    |      |        |      |
  +----------------------+    +----------------------+
        |             |                 |
+------------+ +------------+    +-------------+
|        |   | |        |   |    |        |    |
| 15.564 | A | | 15.565 | B |    | 15.571 | A  |
|        |   | |        |   |    |        |    |
+------------+ +------------+    +-------------+
```

Figure 2B: Instance

In the previous example, accesses were reduced by bringing redundant information upward. Accesses can also be reduced by introducing redundant information within a single level. For example, consider the query: "List all people who received an 'A' in 15.565 and have a GPA exceeding 3.4." If the structure in Figure 1A is used, it would be necessary to perform the lengthy process of computing the GPA for every student who received an 'A' in 15.565. If the structure in Figure 2A is used, the number of accesses is reduced to two per each 15.565 COURSE record. However, the accesses can be reduced even further, to one per 15.565 COURSE record, by repeating the year's GPA in each of the COURSE records for that year, as depicted in Figures 3A and 3B. Although this seems inordinately redundant, it may result in a more efficient system, depending on the frequency of such queries.

These two examples of database restructuring illustrate the general concept of "family redundancy," a technique whereby, "family" information obtained from related records (parents, siblings, children, etc) is introduced into a record in order to improve the performance of certain queries. Note that there is no change in capabilities, i.e., any query that could have been answered before restructuring can be answered at least as efficiently after restructuring. In other words, "family redundancy" will not increase (and may reduce) the number of accesses needed to respond to any query or operation. This is true because "family redundancy" neither alters the basic arrangement of the record types, nor deletes any of the original fields, implying that any query could always be answered by ignoring the newly-inserted information and instead, following the original procedure. Hence, the introduction of family information can never increase the number of accesses, and as was shown in the previous examples, can sometimes reduce them.

Note that two assumptions have been made in the above example. The first is that the database is accessed using a high-level language interface, where each database movement is pre-defined by the programmer. This can be significantly different from the query processor situation where the exact sequence of database operations is unknown to the applications programmer.

The second assumption is that there is no use of virtual data [5]. That is to say, the DBMS cannot store in the GPA field an "equation" based on the GRADE field.

```
STUDENT

              +--------------------------------+
              |            |       |           |
              | ID_NUMBER* | NAME  | ADDRESS   |
              |            |       |           |
              +--------------------------------+
                              |
YEAR                          |
                              |
              +--------------------------------+
              |         |          |           |
              | YEAR*   |   DEPT   |   GPA     |
              |         |          |           |
              +--------------------------------+
                              |
COURSE                        |
                              |
              +--------------------------------+
              |          |         |           |
              | COURSE*  |  GRADE  |   GPA     |
              |          |         |           |
              +--------------------------------+
```

Figure 3A: Repeating GPA in YEAR

```
          +----------------------------------------------+
          |          |                   |               |
          | 123-456  |  Steven Melnick   |   Eastgate    |
          |          |                   |               |
          +----------------------------------------------+
                   |                         |
          +---------------------+    +---------------------+
          |      |        |     |    |      |        |     |
          | 1978 |   6    | 3.5 |    | 1979 |   15   | 4.0 |
          |      |        |     |    |      |        |     |
          +---------------------+    +---------------------+
               |            |                     |
     +------------------+ +------------------+ +------------------+
     |        |   |     | |        |   |     | |        |   |     |
     | 15.564 | A | 3.5 | | 15.565 | B | 3.5 | | 15.571 | A | 4.0 |
     |        |   |     | |        |   |     | |        |   |     |
     +------------------+ +------------------+ +------------------+
```

Figure 3B: Instance

Quantification of Savings Potential Using Family Redundancy

There is a query speed processing advantage and countervailing space and maintenance costs associated with family redundancy:

1) The savings due to faster queries can be computed as follows:

   Ri = difference in cost per query i, before and after database restructuring (function of CPU, I/O, DBMS)

   Ni = average number of queries of type i during time period t

   $$\text{Savings} = \sum_i RiNi \qquad (1)$$

2) The costs of the extra space are:

   b = cost per byte per time period t ($/byte)

   B = number of extra bytes

   $$\text{Cost} = (b * B) \qquad (2)$$

3) The maintenance costs associated with updating the extra records are:

   Uj = cost of updating extra records for data update type j

   Vj = number of data updates of type j per time period t

   $$\text{Cost} = \sum_j UjVj \qquad (3)$$

Therefore, the overall savings per time period t are:

$$\text{Savings}(t) = \left(\sum_i RiNi\right) - (b*B) - \left(\sum_j UjVj\right) \qquad (4)$$

In a later section of this paper, Equation 4 shall be used to predict savings in an actual system.

Improving Performance with Collapsing of Levels

If the structure of Figure 1A was mainly used to answer the query: "Print all the grades for every student," then the total number of records accessed would be the sum of the number of STUDENT, YEAR and COURSE records in the database. If YEAR is collapsed into COURSE as depicted in Figure 4A, then the number of accesses is only the sum of the number of STUDENT and COURSE records.

This example illustrates the concept of "collapsing of levels," whereby, levels of the hierarchy are eliminated and the information that they contained is duplicated in their children. Contrary to family redundancy, collapsing of levels does not always give beneficial results, and in fact, it can give significantly worse results. For example, the query, "What is Melnick's GPA in 1979?" results in having to access all COURSE records in 1978, as well as those of 1979. Before collapsing, one access could be made to the 1979 YEAR record, followed by one to each of its COURSE records.

Since the number of accesses saved per report can be negative, the decision whether or not to collapse levels, is not always clear-cut. Generally, a level can be eliminated if that level has a small number of bytes, and does not contain information that is often used for decision-making in the hierarchy. If the sample hierarchy contained only the year in the YEAR record, and if it was rarely required to pick out a specific year of a student's file (as might be the case with a history file), then there might be a positive benefit in collapsing the YEAR record into the COURSE record. (The inclusion of the GPA is an entirely separate matter.)

```
STUDENT
          +--------------------------------+
          |              |       |         |
          |  ID_NUMBER*  |  NAME | ADDRESS |
          |              |       |         |
          +--------------------------------+
                                 |
COURSE                           |
                                 |
          +--------------------------------------+
          |            |         |       |       |
          |  COURSE*   |  GRADE  | YEAR  | DEPT  |
          |            |         |       |       |
          +--------------------------------------+
```

Figure 4A: Collapsing of levels

```
           +----------------------------------+
           |         |                |         |
           | 123-456 | Steven Melnick | Eastgate |
           |         |                |         |
           +----------------------------------+
                  |                  |       |
    +-----------------------------+  |       |
    |        |   |      |         |  |       |
    | 15.564 | A | 1978 |    6    |  |       |
    |        |   |      |         |  |       |
    +-----------------------------+  |       |
                                     |       |
              +-----------------------------+ |
              |        |   |      |         | |
              | 15.565 | B | 1978 |    6    | |
              |        |   |      |         | |
              +-----------------------------+ |
                                              |
                       +-----------------------------+
                       |        |   |      |         |
                       | 15.571 | A | 1979 |   15    |
                       |        |   |      |         |
                       +-----------------------------+
```

Figure 4B: Instance

Quantification of Savings Potential Using Collapsing of Levels

The savings function for collapsing of levels is identical to Equation 4 with the following exceptions:

1) The savings due to faster queries are the same as before, but Ri can now be negative.

2) Collapsing does not necessarily involve extra storage. The number of extra bytes, B, is determined as follows:

$$\text{Extra Storage} = \{(C*D) - [P(D+D')] - Q\} \qquad (5)$$

where C = number of 'child' records

P = number of 'parent' records

D = number of data bytes per 'parent' record

D' = number of index bytes per 'parent' record

Q = fixed number of bytes required to maintain a level

Case Study

To illustrate the potential for dramatic savings in an actual application, the Birth Defects Information System (BDIS), a joint research project of the March of Dimes Birth Defects Foundation and the Tufts New England Medical Center, will be examined. BDIS provides various computer services to aid physicians in treating patients with birth defects. It is implemented on an IBM/370 under VM/CMS and uses a database management system called FOCUS. This case study centers on the Diagnostic Assist Facility, one service of BDIS, which is still in the testing stage.

The Diagnostic Assist facilty works as follows:

1) Physicians choose from a report those symptoms that describe their patient's condition, e.g., cleft palate, mental retardation. (SELECTION PHASE)
2) Birth defects which exhibit any of the chosen symptoms are retrieved from the database for consideration. (RETRIEVAL PHASE)
3) A score is computed for each of the considered birth defects, based on the strength of the match between the symptoms selected by the physician and those stored in the database for the birth defect. The birth defects are then ranked according to the score, and the top ten are maintained. (SCORING PHASE)
4) The physician is asked questions relating to the top birth defects in order to more carefully refine the diagnosis. The answers are used to re-score the birth defects. (CHALLENGE

PHASE)

For example, if the physician were to enter the following symptoms:

disproportionate short stature

rhizomelia

cleft palate

the computer might respond with the following list of potential syndromes:

Kneist Dysplasia

Hermann-Opitz Arthrogryposis Syndrome

Short Rib-Polydactyly Syndrome

The major storage structure is a hierarchical database which stores the symptoms that are commonly found with each birth defect syndrome. (See Figure 5.) Symptoms which are always seen in patients with that birth defect are considered REQUIRED, others are called non-required or choice symptoms. Closely-related symptoms for a birth defect are grouped into SETs. Associated with each set is a WEIGHT and a CHOICE COUNT. The WEIGHT is the number of points given to the syndrome if the criteria for "satisfying" the set are met. To satisfy a set, the patient's case must match on all of the REQUIRED symptoms in the set, plus some number of the choice symptoms, as indicated by the CHOICE COUNT. The total score for a syndrome is the sum of the weights of the satisfied sets, normalized by the total possible sum for that syndrome. Figure 6 illustrates an instance of the database in table format.

```
                                                              NUMBER OF
SYNDROME                                                     OCCURRENCES
          +----------------------+
          |           |          |
          |  NUMBER*  |   NAME   |                               700
          |           |          |
          +----------------------+
                      |
SET                   |
                      |
      +------------------------------------+
      |            |         |             |
      | SET_NUMBER | WEIGHT  | CHOICE_COUNT|                     2000
      |            |         |             |
      +------------------------------------+
                      |
SYMPTOM               |
                      |
            +--------------------+
            |          |         |
            | SYMPTOM* | REQUIRED|                               13000
            |          |         |
            +--------------------+
```

Figure 5: The BDIS Hierarchy

Number: 123 Name: XYZ

SET	WEIGHT	CHOICE_COUNT	REQUIRED	SYMPTOM
1*	4	2		eye, downward slanting
				eyeball, deepset
				eyeball, atrophy of
				eye, ocular hypertolism
2**	3	1	R	immobile facial expression
				mouth, small
				lips, puckered
				mouth, downturned corners

Figure 6: Instance of the BDIS Hierarchy

---

* In Set 1 of the illustrated syndrome, two of the four symptoms must be matched in order to receive four points. One symptom does not result in any points, while three or four matches still give only four points.

** In Set 2, the patient must have an immobile facial expression, and one of the other three symptoms to receive three points. If the first symptom is not matched, then no points will be awarded (even if all the other symptoms are matched).

The greatest usage of computer time in the BDIS system is during the RETRIEVAL PHASE, when the birth defect information is retrieved from the database. Following is a detailed explanation of the database accesses that are required:

1) Get the first symptom from the physician's list.
2) Do an indexed retrieval to the first occurrence of that SYMPTOM record in the database.
3) Go up to the SYNDROME record to find the number of the birth defect containing that symptom.
4) Get the next occurrence of that SYMPTOM record. (The same symptom may be present in multiple birth defects.) If more exist, return to Step 3.
5) Repeat Steps 2 to 4 for every symptom in the physician's list.
6) For each of the syndromes that has been retrieved (after eliminating duplicates), get all of the associated SET and SYMPTOM records.

The last step is required for the following three reasons:

1) It is not known if all of the required symptoms for a set have been matched, until all of the SYMPTOM records in that set have been retrieved.
2) The total possible score (sum of the set weights) for a birth defect cannot be computed, until all of the SET records for a birth defect are retrieved.
3) Though the SYMPTOM records for sets which have not been selected are not immediately needed, it is more efficient to retrieve all of the SET and SYMPTOM records in one pass, than

to selectively retrieve records. This is due to a facility in FOCUS, whereby multiple records can be retrieved with a single database call. (The effects of this facility on database access costs will be explained in a later section.)

Thus, in order to score the birth defects, the program had to go up the database for each of the occurrences of the selected symptoms. In addition, for each of the potential SYNDROMEs, it was necessary to retrieve all of the associated SET and SYMPTOM records.

Restructuring the BDIS Database

To improve efficiency, the database was restructured with the following changes.

1) The number of required symptoms in a set (REQUIRED COUNT) was inserted into the SET record. This eliminated the need to retrieve all of the SYMPTOM records in a set just to determine if the required symptoms had been matched.
2) The total weight of the sets (TOTAL WEIGHT) was inserted into the SYNDROME record. Now it was no longer necessary to retrieve all of the SET records to determine the maximum possible score.
3) The SET records were incorporated into the SYMPTOM records. This meant that the path up to the SYNDROME record for an indexed symptom was shorter, and that the total number of records under a root SYNDROME record was less.

The first two changes are examples of "family redundancy" techniques that result in the database depicted in Figure 7A. The last change is a collapsing of levels, which gives the database of Figure 7B.

With the database of Figure 7B, a score for each birth defect can be computed by simply going up the database for each occurrence of the selected symptoms. We can then select only the top candidates and retrieve all of their symptoms for use in the CHALLENGE PHASE. As a result, the following reductions in the number of required accesses have been made:

1) The number of accesses required to go up to a root SYNDROME record is two instead of three.

2) The number of trips down the hierarchy is limited to ten (the top birth defects). Formerly, every considered birth defect required a downward trip.

3) Less records are retrieved on each of these downward trips, as the SET records have been collapsed into the SYMPTOM records.

Thus, compared to the original system, a significant reduction in the number of accesses has been made.

```
SYNDROME
                 +------------------------------+
                 |            |      |                |
                 | NUMBER*  | NAME | TOTAL_WEIGHT |
                 |            |      |                |
                 +------------------------------+
                                  |
SET                               |
                                  |
        +---------------------------------------------------+
        |              |         |                |                  |
        | SET_NUMBER | WEIGHT | CHOICE_COUNT | REQUIRED_COUNT |
        |              |         |                |                  |
        +---------------------------------------------------+
                                  |
SYMPTOM                           |
                                  |
                   +--------------------+
                   |            |          |
                   | SYMPTOM* | REQUIRED |
                   |            |          |
                   +--------------------+
```

Figure 7A: BDIS database after Family Redundancy

```
SYNDROME
               +------------------------------+
               |            |      |                |
               | NUMBER*  | NAME | TOTAL_WEIGHT |
               |            |      |                |
               +------------------------------+
                               |
SYMPTOM                        |
                               |
+------------------------------------------------------------------------+
|              |         |                |                  |            |          |
| SET_NUMBER | WEIGHT | CHOICE_COUNT | REQUIRED_COUNT | SYMPTOM* | REQUIRED |
|              |         |                |                  |            |          |
+------------------------------------------------------------------------+
```

Figure 7B: BDIS database after Collapsing of Levels

Quantification of Savings Potential for BDIS

Using Equation 4, and subtracting a further manpower and computer cost for changing the system (K), the following savings equation is obtained:

$$\text{Savings}(t) = (\sum_i RiNi) - (b*B) - (\sum_j UjVj) - K \qquad (6)$$

For this case:

$b*B$	is a minimal cost. The increase in storage was under 250K bytes.
$(\sum_j UjVj)$	is small because the data base is updated only a few times per year. (In test runs, there was no significant difference in the cost of updating the old and new databases.)
i	is 1, i.e., only one type of query (diagnostic assist) exists.
K	is small when averaged over several years of use, and in this particular case, the restructuring coincided with a major overhaul of the system, and thus, did not amount to an additional expense.

Therefore, equation (6) reduces to:

$$\text{Savings}(t) = R * N \qquad (7)$$

R, which is the difference in cost of a query, is exceedingly difficult to compute theoretically. Firstly, FOCUS maintains its data in 4000 byte pages, and therefore, an access may or may not imply an actual disk I/O. Secondly, the cost of retrieving information from the database consists of a relatively fixed charge of making a database call, plus a variable cost depending on the number of records accessed. Lastly, the single call/multiple record facility of FOCUS means that many records can be retrieved at a smaller cost than if they were retrieved independently. Despite all of these problems, and for lack of a better choice, the theoretical estimate of the number of accesses made was used to attempt to predict system costs.

Let us therefore, compute Ao, the number of accesses under the old system, and Ar, the number of accesses under the restructured system.

From the database it was determined that each syndrome contains on the average three sets and 19 symptoms. In addition, the mean number of syndromes in which a particular symptom exists is fifteen. 152 test cases were examined, and each contained an average of eleven symptoms, and retrieved a mean of 210 candidate birth defects.

With this information, Ao can be computed as follows:

```
  Ao               = (Accesses up) + (Accesses down)
  Accesses up     = #Symptoms/Case * #Syndromes/Symptom * #accesses to go up
                   = 495              (11 * 15 * 3)
  Accesses down   = #Candidates   *   #accesses to get all symptoms
                   = 4830             210 * (1 + 3 + 19)
```

Therefore, under the original system, the number of accesses required (Ao) is 5325.

Ar can be computed in a similar manner with the following important exceptions:

1) The number of accesses required to go up to a root SYNDROME record is two instead of three.

2) The number of trips down the hierarchy is limited to the top birth defects. (Due to scoring ties, the average number of trips is twelve rather than ten.)

3) The number of accesses required to get all of the symptoms for a birth defect is 20 instead of 23.

Accesses up = 330	(11 * 15 * 2)
Accesses down = 240	(12 * 20)

The total number of accesses under the new system (Ar) is 570. This implies that the new system will use about one-ninth the number of accesses as the original system.

Experimental Results

In this section, the observed results when the system was implemented are reported. The experiment compared I/O and CPU usage under the original and restructured systems for ten test cases. The cases had from one to twenty-one symptoms.

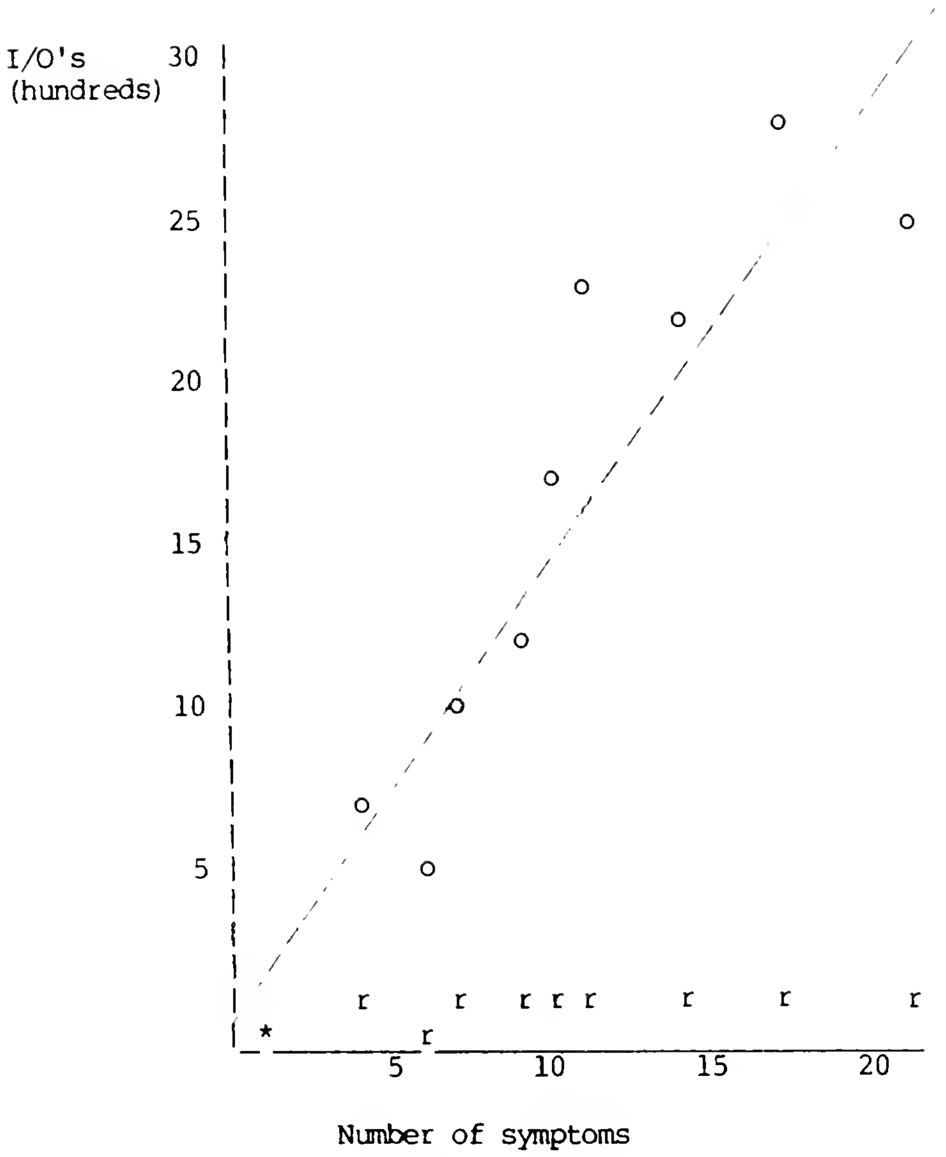
The experimental data is summarized in Figures 8 and 9. In Figure 8, the number of I/O requests is illustrated and in Figure 9, the CPU usage is shown. The data was obtained by monitoring the resource usage for the retrieval phase of the program and subtracting the usage for basic operations, i.e., opening databases, loading modules, etc. Note that since a FOCUS access does not imply a disk I/O, then the absolute values of the experimental results cannot be compared directly with the formulated estimates. However, the following observations can be made:

1) As expected, the I/O usage under the restructured system is relatively constant. This is because no matter how many symptoms are entered, the program only goes down the tree for the top candidate birth defects. On the other hand, the old system went down the hierarchy for every candidate birth defect, and therefore, as more symptoms imply more candidates, its resource usage varies closely with the number of symptoms.

2) The number of I/O's under the restructured system compared to the original system, ranges from one-sixth (with four symptoms) to one twenty-eighth (with seventeen symptoms). (The one symptom case only retrieved five candidate birth defects, and

therefore, both systems had similar usage.)

Using the regression line statistics and the case of eleven symptoms, the new system saves sixteen CPU seconds and 1500 I/O operations. Using a CPU rate of $30 per minute and an I/O rate of $2 per thousand, this amounts to a savings of $11 per diagnosis. With 100 users, each doing five cases per month, the savings could reach $5500 per month.



(o = old system, r = restructured system, * = tie)

(--- = regression line)

Figure 8: Experimental Results (I/O Usage)

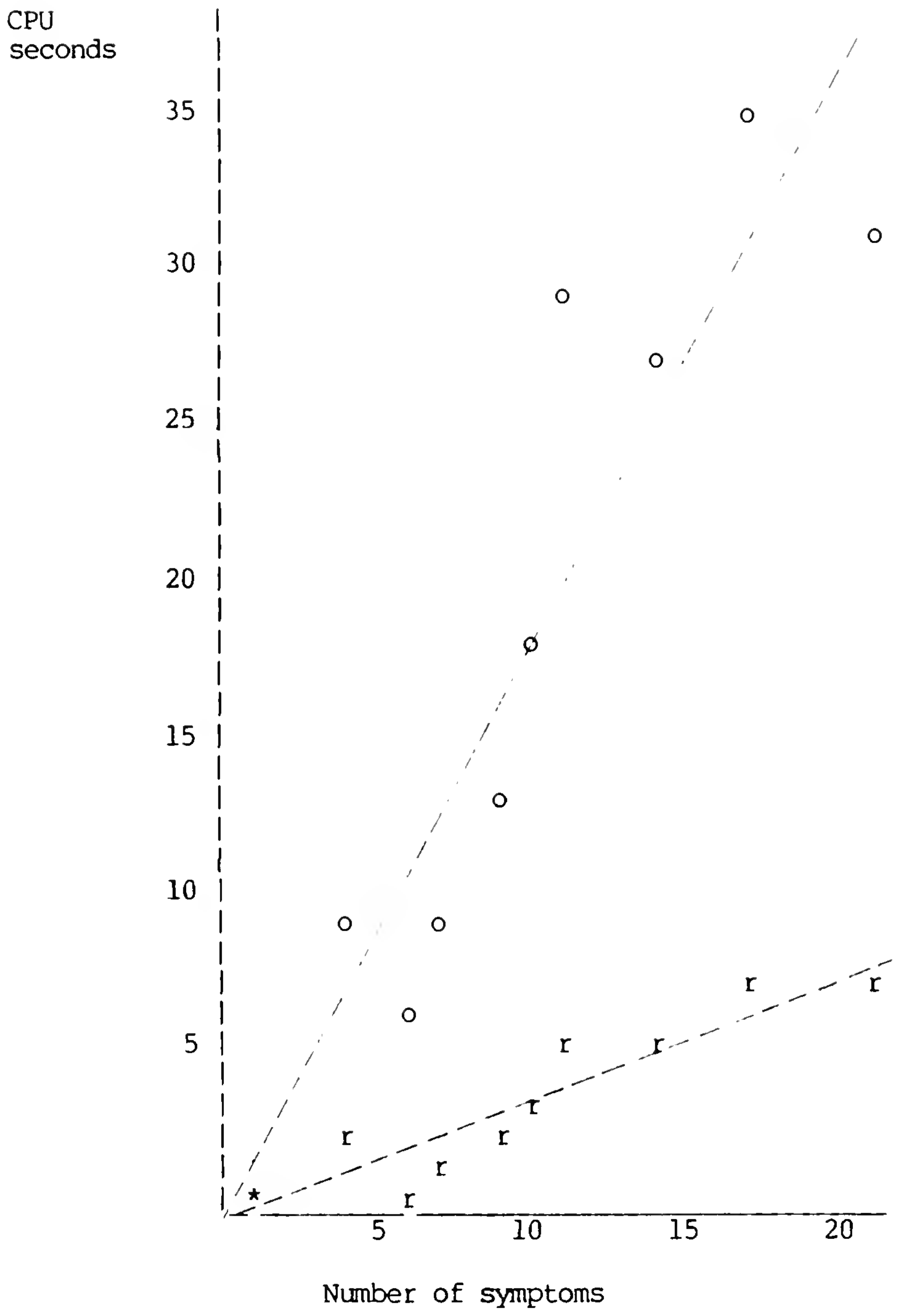


Figure 9: Experimental Results (CPU Usage)

Other Issues to be Considered

Following are some other points of interest that have been touched upon in this paper, but which are beyond the scope of this analysis.

Managerial Implications:

- Often, the costs of the query and the costs of maintenance are borne by two different organizations. Therefore, these costs may be weighed differently by a system implementor.

Technical Issues:

- An interesting question concerns the independence of multiple restructurings of the same database, e.g., are the advantages obtained by collapsing a level independent of other improvements that are being made elsewhere.

Conclusion

The two concepts introduced here, family redundancy and collapsing of levels, can under certain circumstances, lead to dramatic savings. The conditions for a successful database restructuring include:

1) A high ratio of database accesses to updates.
2) A well-defined set of database operations.
3) A small number of data fields that must be made redundant in order to obtain a high reduction in accesses.
4) A system that can be restructured cheaply and efficiently.

References

1. A.F. Cardenas, DATABASE MANAGEMENT SYSTEMS, Allyn and Bacon, Boston, Mass. (1979)

2. A.F. Cardenas, "Analysis and Performance of Inverted Database Structures," COMMUNICATIONS OF THE ACM 18, no. 5, 253-263 (1975)

3. C.J. Date, AN INTRODUCTION TO DATABASE SYSTEMS, 2nd ed., Addison-Wesley, Reading, Mass. (1977)

4. J.J. Donovan and S.E. Madnick, "Institutional and Ad-hoc Decision Support Systems and their Effective Use," DATABASE 8, no. 3, Winter (1977)

5. J.J. Folinus, S.E. Madnick and H.B. Schutzman, "Virtual Information in Data-Base Systems," SLOAN WORKING PAPER 723-74, Center for Information Systems Research, M.I.T., July (1974)

6. IBI, FOCUS USER'S MANUAL, Information Builders Inc., New York, NY (1978)

7. R.G. Ross, "Data Base Systems: Design, Implementation and Management, Part III," COMPUTERWORLD, June 5 (1978)

8. M.E. Senko, E.B. Altman, M.M. Astrahan and P.L. Fehder, "Data Structures and Accessing in Data Base Systems," IBM SYSTEMS JOURNAL 12, no. 1, 30-93 (1974)

9. D.G. Severance, "Some Generalized Modeling Structures for Use in Design of File Organizations," Doctoral Thesis, University of Michigan (1972)

10. G.H. Sockut and R.P. Goldberg, "Database Reorganization - Principles and Practice," ACM COMPUTING SURVEYS 11, no. 4,

371-395 (1979)

11. D.C. Tsichritzis and F.H. Lochovsky, DATABASE MANAGEMENT SYSTEMS, Academic Press, New York, NY (1977)

12. D.C. Tsichritzis and F.H. Lochovsky, "Hierarchical Database Management," ACM COMPUTING SURVEYS 8, no. 1, 105-124 (1976)

13. G. Wiederhold, DATABASE DESIGN, McGraw Hill, New York NY (1977)

HD28.M414 no.1133- 80
Schmalensee, R/The new industrial orga
739568 D*BKS 00132000

3 9080 001 988 648

HD28.M414 no.1134- 80
Banks, Louis. /Living in a newsocracy
740326 D*BKS 00137171

3 9080 002 050 042

HD28.M414 no.1135- 80
Wong, M. Antho/A Monte Carlo study of
740340 D*BKS 00133429

3 9080 002 005 863

HD28.M414 no.1136- 80
Henriksson, Ro/Tests of market timing
740309 D*BKS 00132385

3 9080 001 992 954

HD28.M414 no.1140- 80
Schmalensee, R/Product differentiation
740333 D*BKS 00132649

3 9080 001 996 187

HD28.M414 no.1141- 80
Buer, Terje K./Investigation of make o
740829 D*BKS 00132383

3 9080 001 992 913

HD28.M414 no.1142- 80
Nevis, Edwin C/Gestalt awareness proce
740329 D*BKS 00132363

3 9080 001 992 558

HD28.M414 no.1143- 80
Kolb, Deborah /Onstage with labor medi
740812 D*BKS 00132361

3 9080 001 992 525

HD28.M414 no.1144- 80
Berndt, Ernst /Short run labor product
740816 D*BKS 00132384

3 9080 001 992 939

MIT LIBRARIES

3 9080 004 178 924

1145-80

HD28.M414 no.1146- 80
Wallace, Anne /Getting medical attenti
740836 D*BKS 00132387

3 9080 001 992 996